UTM

UNIVERSITI TEKNOLOGI MALAYSIA

FACULTY OF COMPUTING
SEMESTER II, SESSION 2024/2025

BACHELOR OF COMPUTER SCIENCE (BIOINFORMATICS)

SECB3203 PROGRAMMING FOR BIOINFORMATICS - SECTION 01

PROJECT
ALZHEIMER'S DISEASE PREDICTION

GROUP MEMBERS	MATRIC NO
TAN ZHAO HONG	A23CS0188
CHIN PEI WEN	A23CS0065
KOO XUAN	A23CS0300

LECTURER'S NAME : DR. SEAH CHOON SEN

SUBMISSION DATE : 19 DECEMBER 2025

Table of Contents

1.0 Introduction

Alzheimer's Disease (AD) represents one of the most pressing challenges in modern healthcare, being the most prevalent form of dementia worldwide. As the global population ages, the incidence of AD continues to rise dramatically, affecting millions of individuals and their families. Early and accurate diagnosis of Alzheimer's Disease is critical for timely intervention, treatment planning, and improving patient quality of life.

This project aims to develop a robust machine learning-based classification system for Alzheimer's Disease prediction using comprehensive clinical, demographic, and lifestyle data. By leveraging advanced computational techniques and bioinformatics approaches, we seek to create an effective predictive model that can assist healthcare professionals in early AD detection and risk assessment.

The availability of large-scale healthcare datasets and advancements in machine learning algorithms present unprecedented opportunities to identify patterns and biomarkers associated with Alzheimer's Disease. Through this project, we will explore various machine learning techniques, including feature selection, data preprocessing, model development, and evaluation to create an accurate and interpretable classification system.

1.1 Problem Background

Alzheimer's Disease is a progressive neurodegenerative disorder characterized by cognitive decline, memory loss, and behavioral changes. According to recent studies, AD affects approximately 35.6% of diagnosed dementia cases, with the remaining 64.4% representing individuals without the disease. The challenge lies in the complexity of AD diagnosis, which traditionally relies on:

- **Clinical assessments:** Cognitive and functional tests that may be subjective
- **Medical imaging:** MRI and PET scans that require specialized equipment and expertise
- **Medical history:** Comprehensive patient background that may be incomplete
- **Laboratory tests:** Biomarker analysis that can be expensive and time-consuming

Traditional diagnostic methods often result in:

1. **Late-stage diagnosis:** Many patients are diagnosed only after significant cognitive decline
2. **Inconsistent results:** Variability between different assessment methods and practitioners
3. **Resource constraints:** Limited access to specialized diagnostic facilities, particularly in rural or underserved areas
4. **High costs:** Expensive imaging and laboratory procedures that may not be covered by insurance

The dataset we will utilize contains records from 2,149 patients with 34 comprehensive features spanning multiple domains:

- **Demographic factors:** Age, gender, ethnicity, education level
- **Lifestyle factors:** Physical activity, diet quality, sleep patterns, alcohol consumption, smoking status
- **Medical history:** Family history of AD, cardiovascular disease, diabetes, hypertension, depression
- **Clinical measurements:** BMI, blood pressure, cholesterol levels
- **Cognitive assessments:** MMSE scores, functional assessment scores, memory complaints
- **Symptoms:** Behavioral changes, confusion, disorientation, personality changes

This rich dataset provides an opportunity to develop machine learning models that can identify patterns and risk factors associated with Alzheimer's Disease, potentially enabling earlier detection and intervention.

1.2 Problem Statement

Current Alzheimer's Disease diagnostic approaches face several critical challenges:

1. **Delayed diagnosis:** Most patients are diagnosed in moderate to advanced stages when cognitive decline is already significant, limiting the effectiveness of available interventions
2. **Limited accessibility**: Advanced diagnostic tools such as specialized imaging (PET scans, advanced MRI) and biomarker testing are not readily available in all healthcare settings, particularly in developing regions
3. **High diagnostic costs:** Comprehensive AD assessment can be prohibitively expensive, involving multiple specialist consultations, imaging studies, and laboratory tests
4. **Subjective assessments:** Many cognitive tests rely on clinical judgment, which can vary between practitioners and may be influenced by patient demographics and socioeconomic factors
5. **Data integration challenges:** Patient information is often scattered across multiple sources (clinical notes, lab results, imaging reports), making it difficult to obtain a holistic view for accurate diagnosis
6. **Lack of predictive tools:** Current methods primarily focus on confirming diagnosis rather than predicting risk, missing opportunities for preventive interventions

1.3 Objectives

The primary purpose of this project is to explore how computational and bioinformatics approaches can be applied to analyse clinical datasets and support biomedical research. The project aims :

- To demonstrate practical use of programming tools for data cleaning, data interpretation, and predictive modelling.
- To provide insights into how machine-learning methods may assist in distinguishing between different stages of Alzheimer's disease.
- To cultivate understanding of how computational models contribute to addressing real healthcare challenges.

1.4 Scopes

This project focuses on the application of programming and machine learning techniques in the field of bioinformatics to predict Alzheimer's disease using clinical data. The scope of the study is defined as follows:

Domain of Data
The domain of this study is healthcare and bioinformatics, specifically within the area of neurodegenerative disease analysis. The dataset contains clinical, demographic, lifestyle, and cognitive assessment data related to Alzheimer's disease, enabling computational analysis for medical decision support.

Time Frame of Data
The dataset used in this project is obtained from a publicly available Kaggle repository and represents historical patient records collected over multiple years. The study does not involve real-time or longitudinal tracking of patients, and all analyses are conducted on static, retrospective data.

Type of Data Considered
This study utilizes structured tabular data stored in spreadsheet format (.xlsx, converted to .csv). The data consists of both numerical and categorical variables, including continuous measurements, binary indicators, and ordinal attributes. No unstructured data such as medical images, text records, or genetic sequences are included.

Attributes Used
The attributes used in this study are limited to medically relevant features, including:

- Demographic information (e.g., age, gender, education level)
- Lifestyle factors (e.g., physical activity, diet quality, sleep patterns)

- Medical history (e.g., family history of Alzheimer's disease, cardiovascular disease, diabetes)
- Clinical measurements (e.g., BMI, blood pressure, cholesterol levels)
- Cognitive and functional assessments (e.g., MMSE score, functional assessment score)
- Non-medical identifiers and administrative attributes were excluded to reduce noise and potential bias.

Area of Study

The study lies within the intersection of computer science, bioinformatics, and medical data analysis, with an emphasis on disease prediction and clinical decision support systems. It demonstrates how computational methods can assist in early detection of Alzheimer's disease.

Techniques Used

The project applies several data science and machine learning techniques, including:

- Data preprocessing (missing value handling, normalization, encoding)
- Exploratory Data Analysis (descriptive statistics, grouping, correlation analysis, ANOVA)
- Supervised machine learning classification
- Performance evaluation using statistical and graphical methods

Methodology and Approach

A systematic machine learning pipeline is adopted in this study. The methodology includes data collection, preprocessing, exploratory analysis, feature transformation, model training, evaluation, and comparison. A comparative approach is used to assess multiple classification algorithms to determine the most effective model for Alzheimer's disease prediction.

Testing, Measurement, and Experimental Tools

Model testing and evaluation are performed using:

- Train–test split with an 80:20 ratio
- Cross-validation for performance stability
- Evaluation metrics such as accuracy, precision, recall, F1-score, confusion matrix, and ROC-AUC
- The experiments are conducted using Python libraries such as Pandas, NumPy, Matplotlib, Seaborn, and Scikit-learn.

Test Data

The test data consists of 20% of the original dataset, which is unseen during model training. Stratified sampling is applied to preserve the class distribution of Alzheimer's and non-Alzheimer's cases, ensuring fair and reliable evaluation.

Limits of the Research

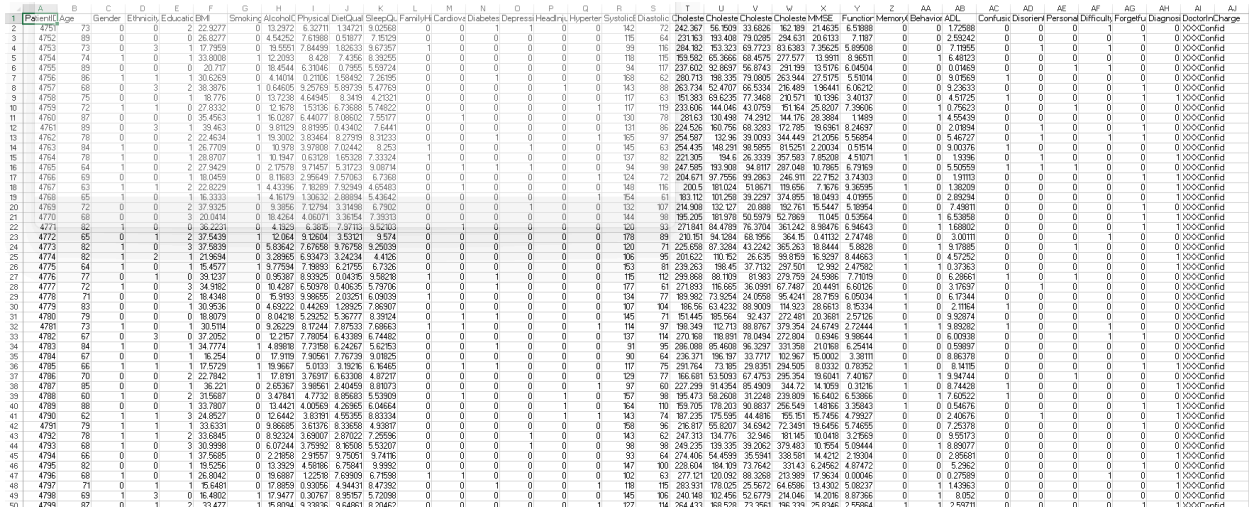
This study has several limitations:

- The dataset is limited to a single publicly available source and may not represent all populations.
- The analysis relies on structured clinical data and does not include imaging, genetic, or biomarker data.
- The models are designed for prediction and risk assessment rather than clinical diagnosis.
- External validation using independent datasets is not performed.
- The results are dependent on data quality and feature availability within the dataset.Gain hands-on experience working with health-related datasets.

2.0 Data Collection and Pre-processing

Our dataset was fetched from Kaggle:
https://www.kaggle.com/datasets/rabieelkharoua/alzheimers-disease-dataset/data

The dataset is in .xlsx, thus we convert it to csv file so that it can be easily read by Python.



PatientID	Age	Gender	Ethnicity	Educatic	BMI	Smoking	Alcohol	Physical	DietQual	SleepQu	FamilyHi	Cardiovs	Diabetes	Depressi	Headinj	Hyperter	Systolic	Diastolic	Choleste	Choleste	Choleste	Choleste	MMSE	Functior	Memory	Behavior	ADL	Confusic	Disorier	Personal	Difficulty	Forgetfu	Diagnosi	DoctorIn	Charge
4751	73	0	0	2	22.9277	0	13.2972	6.32711	1.34721	9.02568	0	0	1	1	0	0	142	72	242.367	56.1509	33.6826	162.189	21.4635	6.51888	0	0	1.72588	0	0	0	1	0	0	XXXConfid		
4752	89	0	0	0	26.8277	0	4.54252	7.61888	0.51877	7.15129	0	0	0	0	0	0	115	64	231.163	193.408	79.1095	294.631	20.6133	7.1187	0	0	2.59242	0	0	0	0	1	0	XXXConfid		
4753	73	0	3	1	17.7959	0	19.5551	7.84499	1.82633	9.67357	1	0	0	0	0	0	99	116	284.182	153.323	69.7723	83.6383	7.35625	5.89508	0	0	7.11955	0	1	0	1	0	0	XXXConfid		
4754	74	1	0	1	33.8008	1	12.2093	8.428	7.4356	8.39255	0	0	0	0	0	0	118	115	159.582	65.3666	68.4575	277.577	13.9911	8.96511	0	1	6.48123	0	0	0	0	0	0	XXXConfid		
4755	89	0	0	0	20.757	0	18.4544	6.31046	0.7955	5.59724	0	0	0	0	0	0	94	117	237.602	92.8697	56.8743	291.198	13.8176	6.04504	0	0	0.01469	0	0	1	1	0	0	XXXConfid		
4756	86	1	1	1	30.6269	0	4.14014	0.21106	1.59492	7.26795	0	0	1	0	0	0	168	62	280.713	198.335	79.0805	263.944	27.5175	5.51614	0	0	9.01569	1	0	0	0	1	0	XXXConfid		
4757	68	0	3	2	38.3876	1	0.64605	9.25769	5.89739	5.47769	0	0	0	0	1	0	143	88	263.734	52.4707	66.5334	216.489	1.96441	6.06212	0	0	9.23633	0	0	0	0	1	0	XXXConfid		
4758	75	0	0	1	16.776	0	13.7238	4.64945	8.3419	4.21321	0	0	0	0	0	0	117	63	151.393	69.6235	77.3468	210.571	10.1396	3.40137	0	0	4.51725	1	0	0	0	1	1	XXXConfid		
4759	72	1	1	0	27.8332	0	12.1679	1.53136	6.73688	5.74622	0	0	0	0	0	1	117	119	233.806	144.046	43.0759	151.64	25.6207	7.38606	0	1	0.75623	0	0	1	0	0	0	XXXConfid		
4760	87	0	0	0	35.4563	1	16.0287	6.44077	8.08602	7.55177	0	1	0	0	0	0	130	78	281.63	130.498	74.2912	144.176	28.3884	1.1489	0	1	4.55439	0	0	0	0	0	0	XXXConfid		
4761	89	0	3	1	39.463	0	9.81129	8.81995	0.43402	7.6441	0	0	0	0	0	0	131	86	224.526	160.756	68.3283	172.795	19.6961	8.24697	0	0	2.01894	0	1	0	0	1	0	XXXConfid		
4762	78	0	0	2	22.4634	1	19.3002	3.83464	8.27919	8.31233	0	0	1	0	0	1	165	97	254.597	132.96	39.6093	344.449	21.2956	5.59264	0	0	5.46727	0	1	0	1	0	1	XXXConfid		
4763	84	1	0	1	26.7709	0	10.978	3.97808	7.02442	9.253	1	0	0	1	0	0	145	63	254.435	148.291	98.5855	815.251	2.20034	0.51514	0	0	9.00376	1	0	0	0	0	0	XXXConfid		
4764	78	1	0	1	28.8707	1	10.1947	0.63128	1.65328	7.33324	1	0	0	0	0	0	137	82	221.305	194.6	26.3339	357.583	7.85208	4.51071	1	0	1.9396	0	1	0	0	0	1	XXXConfid		
4765	64	1	0	2	27.9429	0	2.17578	9.71457	5.31723	9.08714	0	1	1	1	0	0	94	98	247.585	193.908	94.8117	287.049	10.7865	6.79169	0	0	5.50559	1	1	1	0	0	0	XXXConfid		
4766	69	0	0	1	18.0459	0	8.11683	2.95649	7.57063	6.7368	0	0	0	0	0	1	124	72	204.671	97.7556	99.2863	246.911	22.7152	3.74303	0	0	1.91113	0	0	0	0	1	1	XXXConfid		
4767	63	1	1	2	22.8229	1	4.43396	7.18289	7.92949	4.65483	0	1	0	0	0	0	148	116	200.5	181.024	51.8671	119.656	7.1676	9.36595	1	0	1.38209	0	0	0	0	0	1	XXXConfid		
4768	65	1	0	1	16.3333	1	4.16179	1.30632	2.88894	5.43642	0	0	0	0	0	1	154	61	193.112	101.258	39.2297	374.855	18.0493	4.09555	0	0	2.89294	0	0	0	0	0	1	XXXConfid		
4769	72	0	0	2	37.9325	0	9.3856	7.12794	3.11498	6.7902	0	0	0	0	0	1	132	107	214.908	132.127	20.888	192.761	15.5447	5.16954	0	0	7.49611	0	0	0	0	0	1	XXXConfid		
4770	68	0	0	3	20.0414	0	18.4264	4.06071	3.36154	7.39313	0	0	0	0	0	0	144	98	195.205	181.978	50.5979	52.7869	11.045	0.53564	0	1	6.53858	0	0	0	0	1	1	XXXConfid		
4771	82	1	0	0	36.2231	0	4.1329	6.3815	7.9713	9.52303	0	1	0	0	0	0	120	93	271.841	84.4789	76.3704	361.242	8.98476	6.94643	0	1	1.68802	0	0	0	0	1	1	XXXConfid		
4772	65	0	1	2	37.5439	1	12.064	9.12604	3.53121	9.574	0	0	0	0	0	0	178	89	210.151	94.1284	68.1956	364.15	0.41132	2.74748	0	0	3.00111	0	0	0	1	0	0	XXXConfid		
4773	82	1	0	3	37.5839	0	5.63842	7.67688	9.76758	9.25939	0	0	0	0	0	0	120	71	225.658	87.3284	43.2242	365.263	18.8444	5.8828	0	1	9.17985	0	0	0	1	0	0	XXXConfid		
4774	82	1	2	1	21.9694	0	3.28965	6.93473	3.24234	4.4126	0	0	0	0	0	0	106	95	201.622	110.152	26.635	99.8159	16.9297	8.44663	1	0	4.57252	0	0	0	0	0	0	XXXConfid		
4775	64	1	0	1	15.4577	1	9.77594	7.19893	6.21755	6.7326	0	0	0	0	0	0	153	81	239.263	198.45	37.7132	297.501	12.992	2.47582	1	1	0.37363	0	0	0	0	0	0	XXXConfid		
4776	77	0	1	0	39.1237	0	8.95387	8.93925	0.04315	9.58278	1	0	1	0	0	0	115	112	299.868	88.1109	81.983	278.759	24.5986	7.71019	0	0	6.28661	1	0	0	1	0	0	XXXConfid		
4777	72	1	0	3	34.9182	0	10.4287	6.50978	0.40635	5.79706	0	0	1	0	0	0	177	61	271.893	116.665	36.0991	67.7497	20.4491	6.60126	0	0	3.17697	0	1	0	0	0	0	XXXConfid		
4778	71	0	0	2	18.4348	0	15.9193	9.98655	2.03251	6.09039	1	0	0	0	0	0	134	77	189.982	73.9254	24.0598	95.4241	28.7159	6.05034	1	0	6.17344	0	0	0	0	0	0	XXXConfid		
4779	83	0	0	1	30.9536	0	4.83222	0.44269	1.28925	7.86907	0	0	0	0	0	0	107	104	186.56	63.4232	88.9009	114.923	28.6613	8.15334	1	0	2.11164	1	0	0	0	0	0	XXXConfid		
4780	79	0	0	0	18.8879	0	8.04218	5.29252	5.38777	8.39124	0	1	1	0	0	0	145	71	151.445	185.564	92.437	272.481	20.3681	2.57126	0	0	9.52874	0	0	0	0	0	0	XXXConfid		
4781	73	1	0	1	30.5114	0	9.26229	8.17244	7.87533	7.68663	1	1	0	0	0	1	114	97	198.349	112.713	88.8767	379.354	24.6749	2.72444	1	1	9.89282	1	0	0	0	1	0	XXXConfid		
4782	67	0	3	0	37.2052	0	12.2157	7.78054	6.43389	6.74462	0	0	0	0	0	0	137	114	270.168	119.891	78.0494	272.804	0.6946	9.98644	1	0	6.00938	0	0	0	1	1	0	XXXConfid		
4783	84	1	0	1	34.7774	1	4.89898	7.73158	6.24267	5.62153	0	0	1	0	0	0	91	95	286.088	85.4608	96.3297	331.358	21.0168	6.25414	0	0	0.59897	0	0	0	0	0	0	XXXConfid		
4784	67	0	0	1	16.254	0	17.9119	7.90561	7.76739	9.01825	0	0	0	0	0	0	90	64	236.371	196.197	33.7717	102.967	15.0002	3.38111	0	0	8.86378	0	0	0	0	0	0	XXXConfid		
4785	66	1	1	1	17.5729	1	19.9667	5.0133	3.19216	6.16465	0	1	1	0	0	0	117	75	291.764	73.185	29.8351	294.505	8.0332	0.78352	1	0	8.14115	0	0	0	0	1	1	XXXConfid		
4786	70	0	0	2	22.7842	1	17.6191	3.76917	6.63308	4.87217	0	0	0	0	0	0	129	77	166.681	53.5093	67.4753	295.354	19.6041	7.40167	0	1	9.94744	0	0	0	0	1	0	XXXConfid		
4787	85	0	0	1	36.221	0	2.65367	3.98561	2.40459	8.81073	0	0	0	0	0	1	97	60	227.299	91.4354	85.4909	344.72	14.1059	0.31216	1	0	8.74428	1	0	0	0	0	0	XXXConfid		
4788	60	1	0	2	31.5687	0	3.47841	4.7732	8.85683	5.53909	0	1	0	0	1	0	157	98	195.473	58.2608	31.2248	239.809	16.6402	6.53866	0	1	7.60522	1	0	0	0	0	0	XXXConfid		
4789	88	0	0	1	33.7807	0	13.4421	4.00569	4.26965	6.04664	0	0	0	0	1	0	164	110	159.705	176.203	90.8837	256.549	1.48166	3.35843	1	0	0.54676	0	0	0	0	1	0	XXXConfid		
4790	62	1	1	3	24.8527	0	12.6442	3.63191	4.55395	8.83334	0	0	0	0	1	1	143	74	187.235	175.595	44.4816	155.151	15.7456	4.79827	0	0	2.40676	0	1	0	0	0	0	XXXConfid		
4791	79	1	1	1	33.6331	0	9.86685	3.61376	8.33658	4.93817	0	0	0	0	0	0	158	96	216.817	55.8207	34.6942	72.3491	19.6456	5.74655	0	0	7.25378	0	0	0	0	1	0	XXXConfid		
4792	78	1	1	2	33.6945	0	8.92324	3.69007	2.87022	7.25596	0	0	0	1	0	0	143	62	247.313	134.776	32.946	181.145	10.0418	3.21569	0	0	9.55173	0	0	0	0	0	0	XXXConfid		
4793	68	1	0	3	30.9898	0	8.07244	3.75982	8.9598	5.53207	0	0	0	1	0	0	98	98	249.235	139.335	39.2062	378.493	10.1854	5.69444	0	1	8.89077	0	0	0	0	0	0	XXXConfid		
4794	66	0	0	1	37.5685	0	2.21858	2.91557	9.75051	9.74116	0	0	0	0	0	0	93	64	274.405	54.4599	35.5941	338.581	14.4212	2.19304	0	0	2.85681	0	0	0	0	0	0	XXXConfid		
4795	82	0	0	1	19.5256	0	13.3929	4.58186	6.75841	9.9992	0	0	0	0	0	0	147	100	228.604	184.109	73.7642	331.43	6.24562	4.87472	0	0	5.2962	0	0	0	0	1	0	XXXConfid		
4796	68	1	0	1	26.8042	0	19.6887	1.22518	7.69909	6.71598	1	1	0	0	0	0	102	63	277.121	120.092	88.3268	213.989	17.9634	0.00046	0	0	0.27589	0	0	0	1	0	0	XXXConfid		
4797	71	0	1	1	15.6481	0	17.9859	0.93056	4.94431	8.47392	0	0	1	0	0	1	118	115	283.931	178.025	25.5672	64.6586	13.4302	5.68237	0	1	1.43963	0	0	0	0	0	0	XXXConfid		
4798	69	1	3	0	16.4802	1	17.9477	0.30767	8.95157	5.72098	0	0	0	0	0	0	145	106	240.148	102.456	52.6779	214.046	14.2016	8.87366	0	1	8.052	0	0	0	0	0	0	XXXConfid		
4799	87	0	1	2	33.477	1	15.8094	9.33836	9.64861	8.20462	0	0	0	0	0	1	122	114	264.433	168.528	73.3561	196.339	25.8346	2.55864	1	1	2.59711	0	0	0	0	1	0	XXXConfid		

2.1 Importing Dataset

- Python packages for Data Science

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, confusion_matrix, classification_report,
                             roc_auc_score, roc_curve)
# from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
import warnings
warnings.filterwarnings('ignore')
```

- pandas: Data manipulation and analysis
- numpy: Numerical operations and array handling
- matplotlib: Data visualization
- seaborn: Statistical data visualization
- scikit-learn: Machine learning algorithms and preprocessing

- Importing Dataset File

```python
# ================================================================================
# 1. DATA LOADING
# ================================================================================

def load_data(filepath):
    """Load the Alzheimer's disease dataset"""
    df = pd.read_csv(filepath)
    print(f"Dataset shape: {df.shape}")
    print(df)
```

```python
if __name__ == "__main__":
    FILEPATH = "alzheimers_disease_data.csv"
    TARGET_COLUMN = 'Diagnosis'
```

```
======================================================================
ALZHEIMER'S DISEASE PREDICTION - ML PIPELINE
======================================================================

[1] Loading Data...
Dataset shape: (2149, 35)
      PatientID  Age  Gender  Ethnicity  EducationLevel        BMI  ...  Disorientation  PersonalityChanges  DifficultyCompletingTasks  Forgetfulness  Diagnosis  DoctorInCharge
0          4751   73       0          0               2  22.927749  ...               0                   0                          1              0          0       XXXConfid
1          4752   89       0          0               0  26.827681  ...               0                   0                          0              1          0       XXXConfid
2          4753   73       0          3               1  17.795882  ...               1                   0                          1              0          0       XXXConfid
3          4754   74       1          0               1  33.800817  ...               0                   0                          0              0          0       XXXConfid
4          4755   89       0          0               0  20.716974  ...               0                   1                          1              0          0       XXXConfid
...         ...  ...     ...        ...             ...        ...  ...             ...                 ...                        ...            ...        ...             ...
2144       6895   61       0          0               1  39.121757  ...               0                   0                          0              0          1       XXXConfid
2145       6896   75       0          0               2  17.857903  ...               0                   0                          0              0          1       XXXConfid
2146       6897   77       0          0               1  15.476479  ...               0                   0                          0              0          1       XXXConfid
2147       6898   78       1          3               1  15.299911  ...               0                   0                          0              1          1       XXXConfid
2148       6899   72       0          0               2  33.289738  ...               1                   0                          0              1          0       XXXConfid

[2149 rows x 35 columns]
```

These screenshots show the initial dataset loading process.

- Dataset Information

```
46      print(f"\nDataset info:")
47      print(df.info())
```

```
Dataset info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2149 entries, 0 to 2148
Data columns (total 35 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   PatientID                  2149 non-null   int64
 1   Age                        2149 non-null   int64
 2   Gender                     2149 non-null   int64
 3   Ethnicity                  2149 non-null   int64
 4   EducationLevel             2149 non-null   int64
 5   BMI                        2149 non-null   float64
 6   Smoking                    2149 non-null   int64
 7   AlcoholConsumption         2149 non-null   float64
 8   PhysicalActivity           2149 non-null   float64
 9   DietQuality                2149 non-null   float64
 10  SleepQuality               2149 non-null   float64
 11  FamilyHistoryAlzheimers    2149 non-null   int64
 12  CardiovascularDisease      2149 non-null   int64
 13  Diabetes                   2149 non-null   int64
 14  Depression                 2149 non-null   int64
 15  HeadInjury                 2149 non-null   int64
 16  Hypertension               2149 non-null   int64
 17  SystolicBP                 2149 non-null   int64
 18  DiastolicBP                2149 non-null   int64
 19  CholesterolTotal           2149 non-null   float64
 20  CholesterolLDL             2149 non-null   float64
```

```
 21  CholesterolHDL             2149 non-null   float64
 22  CholesterolTriglycerides   2149 non-null   float64
 23  MMSE                       2149 non-null   float64
 24  FunctionalAssessment       2149 non-null   float64
 25  MemoryComplaints           2149 non-null   int64
 26  BehavioralProblems         2149 non-null   int64
 27  ADL                        2149 non-null   float64
 28  Confusion                  2149 non-null   int64
 29  Disorientation             2149 non-null   int64
 30  PersonalityChanges         2149 non-null   int64
 31  DifficultyCompletingTasks  2149 non-null   int64
 32  Forgetfulness              2149 non-null   int64
 33  Diagnosis                  2149 non-null   int64
 34  DoctorInCharge             2149 non-null   object
dtypes: float64(12), int64(22), object(1)
memory usage: 587.7+ KB
None
```

These screenshots show the dataset information details.

- Feature Removal

```python
# ============================================================================
# FEATURES TO REMOVE (Non-medical and weak evidence features)
# ============================================================================
FEATURES_TO_REMOVE = [
    'PatientID',           # Administrative identifier
    'DoctorInCharge',      # Administrative data
    'Gender',              # Weak predictor
    'Ethnicity',           # May introduce bias, weak medical relevance
    'AlcoholConsumption',  # Inconsistent evidence
    'SleepQuality'         # Weak/inconsistent evidence
]

```

```python
    # Remove non-medical and weak evidence features
    print(f"\n{'='*80}")
    print("REMOVING NON-MEDICAL AND WEAK EVIDENCE FEATURES")
    print(f"{'='*80}")
    features_removed = [col for col in FEATURES_TO_REMOVE if col in df.columns]
    if features_removed:
        print(f"\nRemoving features: {features_removed}")
        df = df.drop(columns=features_removed)
        print(f"New dataset shape: {df.shape}")
    else:
        print("\nNo features to remove (features not found in dataset)")

    return df
```

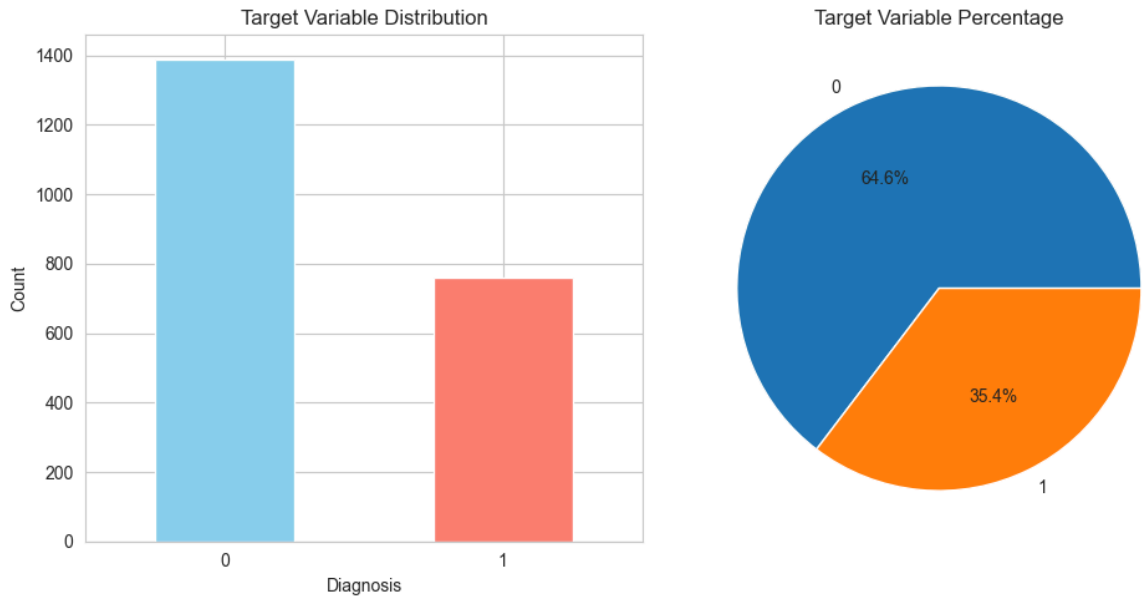
```
================================================================================
REMOVING NON-MEDICAL AND WEAK EVIDENCE FEATURES
================================================================================

Removing features: ['PatientID', 'DoctorInCharge', 'Gender', 'Ethnicity', 'AlcoholConsumption', 'SleepQuality']
New dataset shape: (2149, 29)
```

These screenshots show the removal of non-medical and weak-evidence features such as patient identifiers and administrative attributes. These features were excluded to reduce noise, prevent bias, and ensure that the model focuses only on medically relevant variables that contribute meaningfully to Alzheimer's disease prediction.

2.2 Data wrangling

- Target Variable Distribution

```python
def preprocess_data(df, target_column='Diagnosis'):
    """Preprocess the data for machine learning"""

    # Separate features and target
    X = df.drop(columns=[target_column])
    y = df[target_column]

    # Encode target variable if it's categorical
    if y.dtype == 'object':
        print(f"\nEncoding target variable")
        le_target = LabelEncoder()
        y = le_target.fit_transform(y)
```

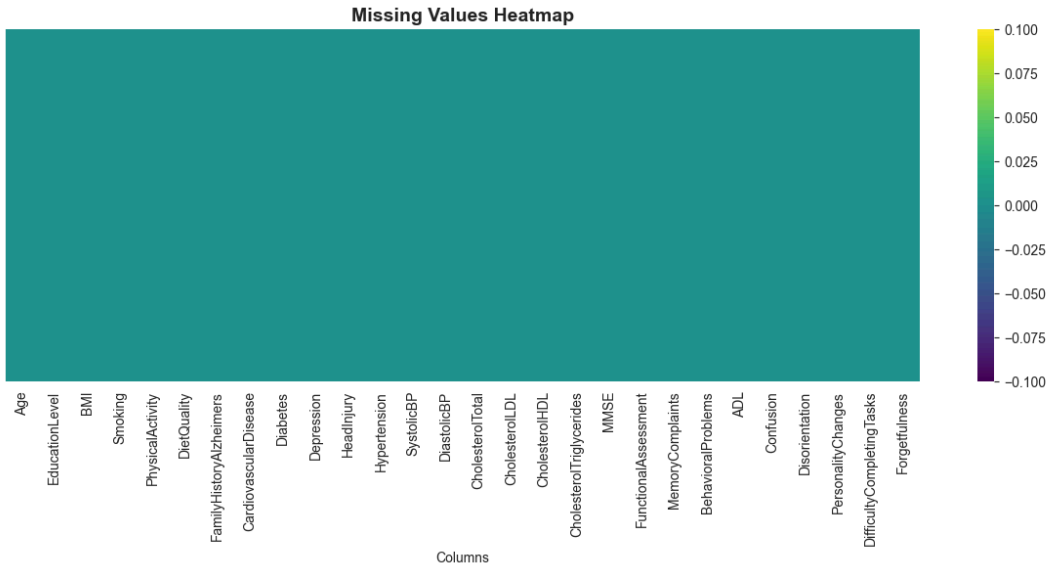


This figure illustrates the distribution of the target variable (Diagnosis) using both bar and pie charts. Visualizing the class distribution helps identify potential class imbalance and confirms that the problem is a binary classification task, which is suitable for the selected machine learning models.

- Identifying and handling missing values

```python
# VISUALIZE MISSING VALUES
print("\n--- Identifying and Handling Missing Values ---")
print("\n1. Missing Values Count:")
print(X.isnull().sum())

print("\n2. Visualizing Missing Values:")
plt.figure(figsize=(12, 6))
sns.heatmap(X.isnull(), cbar=True, cmap='viridis', yticklabels=False)
plt.title('Missing Values Heatmap', fontsize=14, fontweight='bold')
plt.xlabel('Columns')
plt.tight_layout()
plt.savefig('missing_values_heatmap.png', dpi=300)
plt.show()

# Handle missing values
if X.isnull().sum().sum() > 0:
    print("\n3. Handling missing values...")
    X = X.fillna(X.median())
    print("Missing values after handling:")
    print(X.isnull().sum())
else:
    print("\n3. No missing values detected!")
```

```
2. Visualizing Missing Values:

3. No missing values detected!
```



These screenshots display the detection of missing values across all input features. Median imputation was applied to numerical attributes to handle missing data, as it is robust to outliers and preserves the overall distribution of the dataset. This step ensures data completeness before further processing.

- Data Formatting

```python
# DATA FORMATTING
print("\n--- Data Formatting ---")
print("\n1. Current Data Types:")
print(X.dtypes)

print("\n2. Numerical Columns:")
numerical_cols = X.select_dtypes(include=[np.number]).columns
print(list(numerical_cols))

print("\n3. Categorical Columns:")
categorical_cols_orig = X.select_dtypes(include=['object']).columns
print(list(categorical_cols_orig))
```

```
--- Data Formatting ---

1. Current Data Types:
Age                        int64
EducationLevel             int64
BMI                      float64
Smoking                    int64
PhysicalActivity         float64
DietQuality              float64
FamilyHistoryAlzheimers    int64
CardiovascularDisease      int64
Diabetes                   int64
Depression                 int64
HeadInjury                 int64
Hypertension               int64
SystolicBP                 int64
DiastolicBP                int64
CholesterolTotal         float64
CholesterolLDL           float64
CholesterolHDL           float64
CholesterolTriglycerides float64
MMSE                     float64
FunctionalAssessment     float64
MemoryComplaints           int64
BehavioralProblems         int64
ADL                      float64
Confusion                  int64
Disorientation             int64
PersonalityChanges         int64
DifficultyCompletingTasks  int64
Forgetfulness              int64
dtype: object

2. Numerical Columns:
['Age', 'EducationLevel', 'BMI', 'Smoking', 'PhysicalActivity', 'DietQuality', 'FamilyHistoryAlzheimers', 'CardiovascularDisease', 'Diabetes', 'Depression', 'HeadInjury', 'Hypert
ension', 'SystolicBP', 'DiastolicBP', 'CholesterolTotal', 'CholesterolLDL', 'CholesterolHDL', 'CholesterolTriglycerides', 'MMSE', 'FunctionalAssessment', 'MemoryComplaints', 'Beh
avioralProblems', 'ADL', 'Confusion', 'Disorientation', 'PersonalityChanges', 'DifficultyCompletingTasks', 'Forgetfulness']

3. Categorical Columns:
[]
```

This step verifies and formats the dataset by inspecting data types and separating numerical and categorical features. Ensuring correct data types is essential before applying normalization, binning, and encoding, as machine learning algorithms require numerical and consistently formatted inputs.

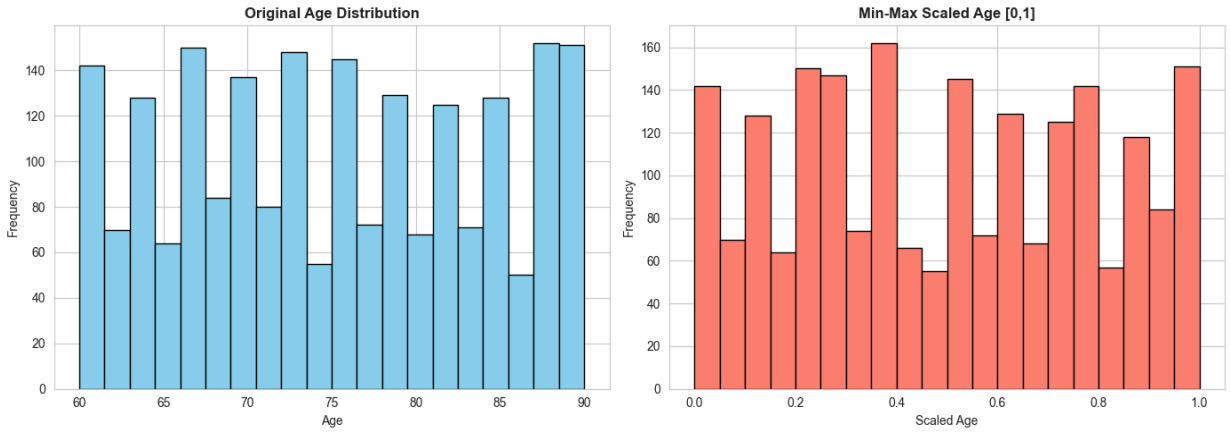
- Data Normalization

```python
# DATA NORMALIZATION
print("\n--- Data Normalization ---")

# Import MinMaxScaler
from sklearn.preprocessing import MinMaxScaler

print("\n1. Min-Max Scaling (Normalization)")
print("   Formula: x_scaled = (x - x_min) / (x_max - x_min)")
print("   Result: Values scaled to range [0, 1]")

if 'Age' in X.columns:
    # Create a copy for demonstration
    min_max_scaler = MinMaxScaler()
    age_minmax = min_max_scaler.fit_transform(X[['Age']])

    print("\n   Example: Age Column")
    print(f"   Original range: [{X['Age'].min():.2f}, {
    print(f"   Scaled range: [{age_minmax.min():.2f}, {age_minmax.max():.2f}]")
```

(variable) age_minmax: ndarray[_AnyShape, dtype[Any]]

```python
    # Show comparison
    comparison_df = pd.DataFrame({
        'Original_Age': X['Age'].head(10),
        'MinMax_Scaled': age_minmax[:10].flatten()
    })
    print("\n   Sample comparison:")
    print(comparison_df)

    # Visualize
    fig, axes = plt.subplots(1, 2, figsize=(14, 5))
    axes[0].hist(X['Age'], bins=20, color='skyblue', edgecolor='black')
    axes[0].set_title('Original Age Distribution', fontsize=12, fontweight='bold')
    axes[0].set_xlabel('Age')
    axes[0].set_ylabel('Frequency')

    axes[1].hist(age_minmax, bins=20, color='salmon', edgecolor='black')
    axes[1].set_title('Min-Max Scaled Age [0,1]', fontsize=12, fontweight='bold')
    axes[1].set_xlabel('Scaled Age')
    axes[1].set_ylabel('Frequency')

    plt.tight_layout()
    plt.savefig('minmax_scaling.png', dpi=300)
    plt.show()

print("\n2. Z-Score Normalization (Standardization)")
print("   Formula: z = (x - μ) / σ")
print("   Note: This will be applied later using StandardScaler")
```

```
--- Data Normalization ---

1. Min-Max Scaling (Normalization)
   Formula: x_scaled = (x - x_min) / (x_max - x_min)
   Result: Values scaled to range [0, 1]

   Example: Age Column
   Original range: [60.00, 90.00]
   Scaled range: [0.00, 1.00]

   Sample comparison:
   Original_Age  MinMax_Scaled
0            73       0.433333
1            89       0.966667
2            73       0.433333
3            74       0.466667
4            89       0.966667
5            86       0.866667
6            68       0.266667
7            75       0.500000
8            72       0.400000
9            87       0.900000

2. Z-Score Normalization (Standardization)
   Formula: z = (x - μ) / σ
   Note: This will be applied later using StandardScaler
```

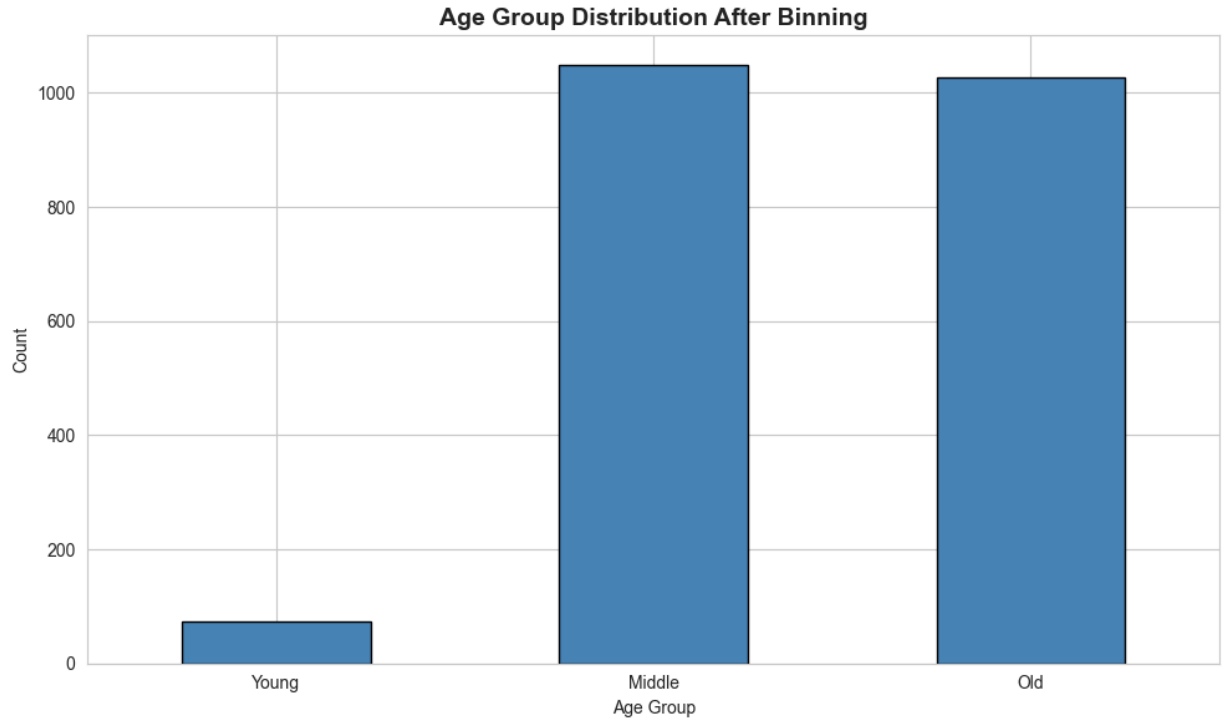


This output demonstrates Min–Max normalization applied to the Age feature, scaling values into the range [0,1]. The comparison between original and scaled values highlights how normalization prevents features with larger numerical ranges from dominating the learning process of distance-based and gradient-based models.

- Binning

```python
# BINNING
print("\n--- Binning ---")

if 'Age' in X.columns:
    print("\n1. Age Binning:")
    X['Age_binned'] = pd.cut(
        X['Age'],
        bins=[0, 60, 75, 100],
        labels=['Young', 'Middle', 'Old']
    )

    # Show binning results
    print("\n   Sample of Age Binning:")
    print(X[['Age', 'Age_binned']].head(10))

    print("\n   Age Group Distribution:")
    print(X['Age_binned'].value_counts())

    # Visualize
    plt.figure(figsize=(10, 6))
    X['Age_binned'].value_counts().sort_index().plot(kind='bar', color='steelblue', edgecolor='black')
    plt.title('Age Group Distribution After Binning', fontsize=14, fontweight='bold')
    plt.xlabel('Age Group')
    plt.ylabel('Count')
    plt.xticks(rotation=0)
    plt.tight_layout()
    plt.savefig('age_binning.png', dpi=300)
    plt.show()
```

```
--- Binning ---

1. Age Binning:

   Sample of Age Binning:
   Age Age_binned
0   73     Middle
1   89        Old
2   73     Middle
3   74     Middle
4   89        Old
5   86        Old
6   68     Middle
7   75     Middle
8   72     Middle
9   87        Old

   Age Group Distribution:
Age_binned
Middle    1048
Old       1027
Young       74
Name: count, dtype: int64
```



-

In this step, the continuous Age feature was transformed into categorical age groups using binning. This approach helps capture non-linear patterns related to age and Alzheimer's risk, improves interpretability, and allows the model to learn differences between broader age groups rather than relying only on exact numerical values.

- Indicator variables

```python
# INDICATOR VARIABLES
print("\n--- Indicator Variables (One-Hot Encoding) ---")

categorical_cols = X.select_dtypes(include=['object', 'category']).columns

if len(categorical_cols) > 0:
    print(f"\n1. Categorical columns to encode: {list(categorical_cols)}")

    # Show before encoding
    print("\n2. Before encoding (first 5 rows):")
    print(X[categorical_cols].head())

    # Apply one-hot encoding
    X = pd.get_dummies(X, columns=categorical_cols, drop_first=True)

    print(f"\n3. After one-hot encoding:")
    print(f"   Original features: {len(categorical_cols)}")
    print(f"   New indicator columns created: {len([col for col in X.columns if any(cat in col for cat in categorical_cols)])}")
    print(f"   Total columns now: {X.shape[1]}")
```

```
--- Indicator Variables (One-Hot Encoding) ---

1. Categorical columns to encode: ['Age_binned']

2. Before encoding (first 5 rows):
  Age_binned
0     Middle
1        Old
2     Middle
3     Middle
4        Old

3. After one-hot encoding:
   Original features: 1
   New indicator columns created: 2
   Total columns now: 30
```

These screenshots show the conversion of categorical features created through binning into numerical indicator variables using one-hot encoding. This transformation is necessary because machine learning models require numerical inputs and cannot directly process categorical text labels.

- Train–Test Split and Z-Score Normalization

```python

    # SPLIT AND SCALE
    # Split the data
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y
    )

    # Scale the features (Z-score normalization)
    print("\n--- Final Scaling (Z-Score Normalization) ---")
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    print(f"\nTraining set size: {X_train_scaled.shape}")
    print(f"Test set size: {X_test_scaled.shape}")

    print("\n" + "="*80)
    print("DATA PREPROCESSING COMPLETE")
    print("="*80)

    return X_train_scaled, X_test_scaled, y_train, y_test, X.columns

```

```
--- Final Scaling (Z-Score Normalization) ---

Training set size: (1719, 30)
Test set size: (430, 30)

================================================================================
DATA PREPROCESSING COMPLETE
================================================================================
```

The dataset was split into training and testing sets using stratified sampling to preserve class distribution. Z-score normalization was then applied using StandardScaler to standardize features, ensuring that all variables contribute equally during model training.

2.3 Software and Hardware Requirements

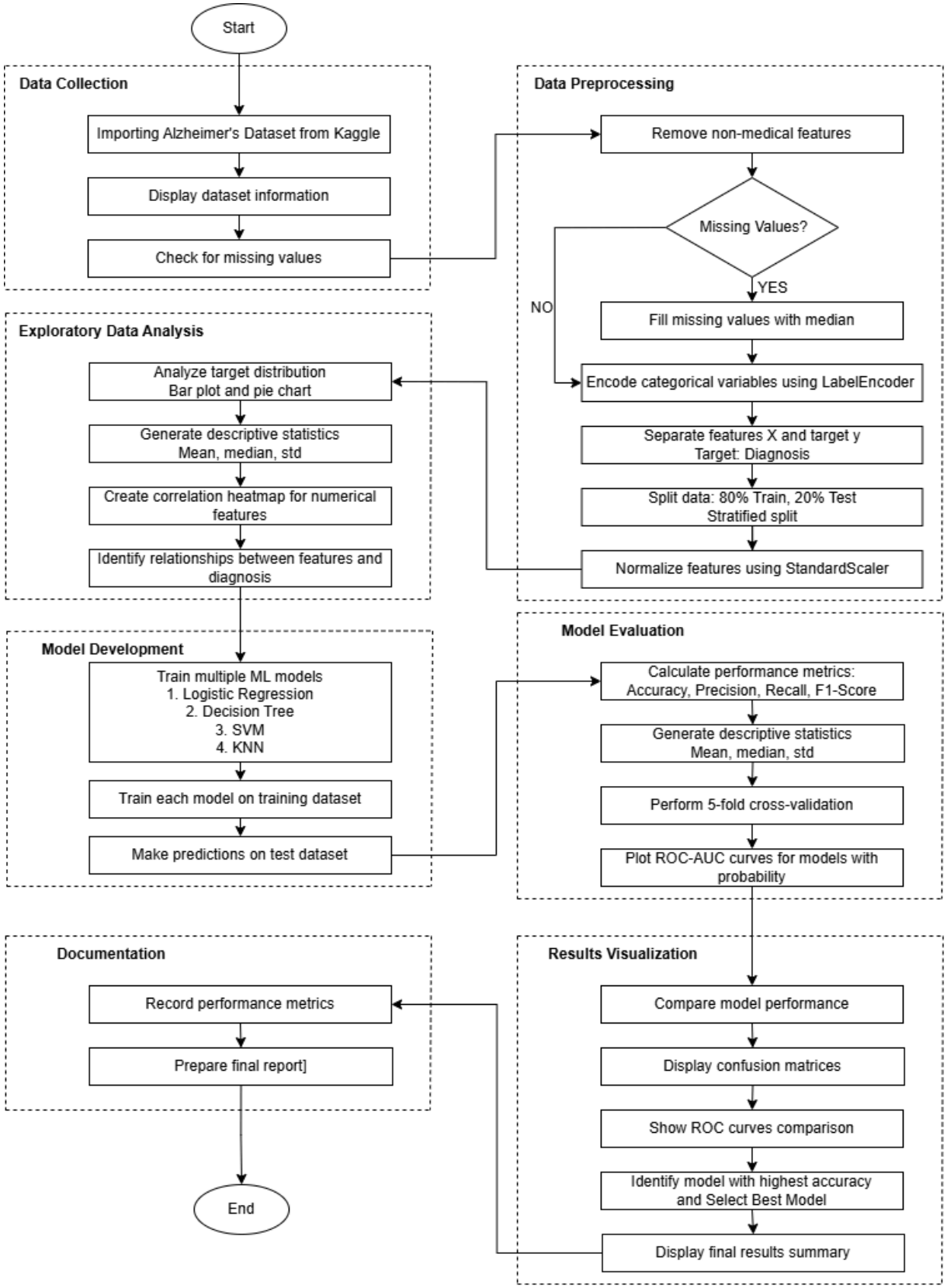
2.3.1 Software Tools

- Visual Studio Code
    - Integrated Development Environment (IDE) for script development and project organization
- Python
    - Programming language used for data preprocessing, analysis, and machine learning model development.
- Jupyter Notebook
    - Environment for code execution, data visualization, and reproducibility of experiments.
- Python libraries
    - Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn for data manipulation, visualization, and machine learning tasks

2.3.2 Hardware Specifications

- Computer
    - System type:64-bit operating system
    - Processor: 11th Gen Intel(R) Core(TM) i5-11300H @ 3.10GHz
    - Memory (RAM): 8GB, sufficient for handling datasets and training classical machine learning models
    - Storage: 476GB, adequate for storing datasets, model parameters, and project files

3.0 Flowchart of the Proposed Approach



3.1 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is an essential step in understanding the structure, characteristics, and patterns within the dataset before building machine learning models. In this project, EDA is conducted using Pandas, NumPy, Matplotlib, and Seaborn to explore the Alzheimer's Disease dataset. The analysis focuses on descriptive statistics, data grouping, analysis of variance (ANOVA), and correlation analysis.

3.1.1 Descriptive Statistics

Descriptive statistics are used to summarize the main characteristics of the dataset numerically. Using the df.describe() function from Pandas, statistical measures such as mean, standard deviation, minimum, maximum, and quartiles are calculated for all numerical attributes.

```
101     # Statistical summary
102     print("\nStatistical Summary:")
103     print(df.describe())
104
105     return df
```

```
[2] Performing Exploratory Data Analysis...

Statistical Summary:
              Age  EducationLevel          BMI      Smoking  PhysicalActivity  ...  Disorientation  PersonalityChanges  DifficultyCompletingTasks  Forgetfulness     Diagnosis
count  2149.000000     2149.000000  2149.000000  2149.000000       2149.000000  ...     2149.000000         2149.000000                2149.000000    2149.000000  2149.000000
mean     74.908795        1.286645    27.655697     0.288506          4.920202  ...        0.158213            0.150768                   0.158678       0.301536     0.353653
std       8.990221        0.904527     7.217438     0.453173          2.857191  ...        0.365026            0.357906                   0.365461       0.459032     0.478214
min      60.000000        0.000000    15.008851     0.000000          0.003616  ...        0.000000            0.000000                   0.000000       0.000000     0.000000
25%      67.000000        1.000000    21.611408     0.000000          2.570626  ...        0.000000            0.000000                   0.000000       0.000000     0.000000
50%      75.000000        1.000000    27.823924     0.000000          4.766424  ...        0.000000            0.000000                   0.000000       0.000000     0.000000
75%      83.000000        2.000000    33.869778     1.000000          7.427899  ...        0.000000            0.000000                   0.000000       1.000000     1.000000
max      90.000000        3.000000    39.992767     1.000000          9.987429  ...        1.000000            1.000000                   1.000000       1.000000     1.000000

[8 rows x 29 columns]
```

3.1.2 Basic of Grouping

Grouping analysis is performed using the groupby() function to examine how different features behave across groups.

Age Grouping

The Age attribute is grouped into three categories:

Young (≤ 60 years)

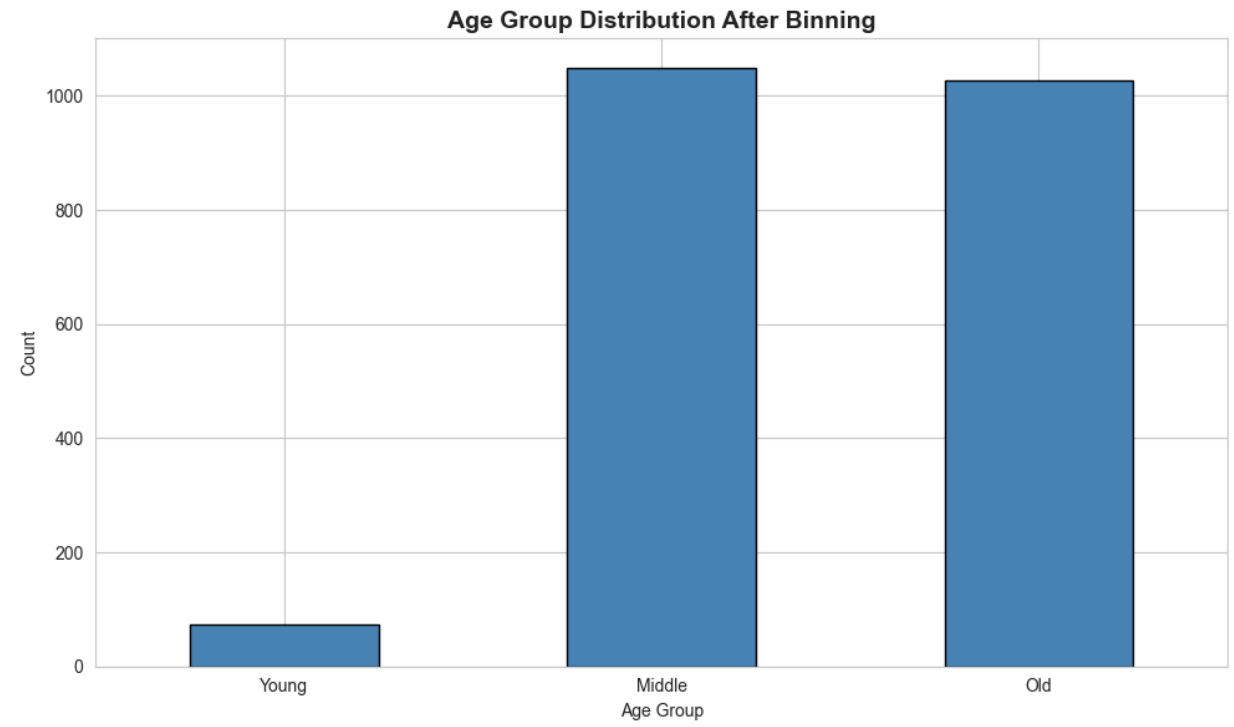
Middle-aged (61–75 years)

Old (> 75 years)

This binning approach helps simplify analysis and reveals trends related to age categories. The frequency of each age group is visualized using bar charts, allowing comparison of population distribution across age ranges.

Grouping helps highlight how different demographic segments contribute to Alzheimer's diagnosis patterns.

```python
if 'Age' in X.columns:
    print("\n1. Age Binning:")
    X['Age_binned'] = pd.cut(
        X['Age'],
        bins=[0, 60, 75, 100],
        labels=['Young', 'Middle', 'Old']
    )

    # Show binning results
    print("\n   Sample of Age Binning:")
    print(X[['Age', 'Age_binned']].head(10))

    print("\n   Age Group Distribution:")
    print(X['Age_binned'].value_counts())

    # Visualize
    plt.figure(figsize=(10, 6))
    X['Age_binned'].value_counts().sort_index().plot(kind='bar', color='steelblue', edgecolor='black')
    plt.title('Age Group Distribution After Binning', fontsize=14, fontweight='bold')
    plt.xlabel('Age Group')
    plt.ylabel('Count')
    plt.xticks(rotation=0)
    plt.tight_layout()
    plt.savefig('age_binning.png', dpi=300)
    plt.show()
```



3.1.3 ANOVA

Analysis of Variance (ANOVA) is applied to determine whether there are statistically significant differences between numerical features across different diagnosis groups.
ANOVA evaluates:
Whether the mean values of numerical features differ significantly between Alzheimer's and non-Alzheimer's patients.
If observed differences are due to actual group effects rather than random variation.
This step supports feature relevance assessment, helping identify which attributes are more influential in distinguishing diagnosis outcomes.

```
106     from scipy.stats import f_oneway
107
108     print("\n--- ANOVA Test (Age vs Diagnosis) ---")
109
110     group_0 = df[df['Diagnosis'] == 0]['Age']
111     group_1 = df[df['Diagnosis'] == 1]['Age']
112
113     f_stat, p_value = f_oneway(group_0, group_1)
114
115     print(f"F-statistic: {f_stat}")
116     print(f"P-value: {p_value}")
```

```
--- ANOVA Test (Age vs Diagnosis) ---
F-statistic: 0.06467450176025781
P-value: 0.799279022412292
```

3.1.4 Correlation

Correlation analysis is conducted using the Pearson correlation coefficient through the corr() function in Pandas.

Correlation Matrix
A correlation matrix is generated for all numerical features to measure the strength and direction of linear relationships between variables.

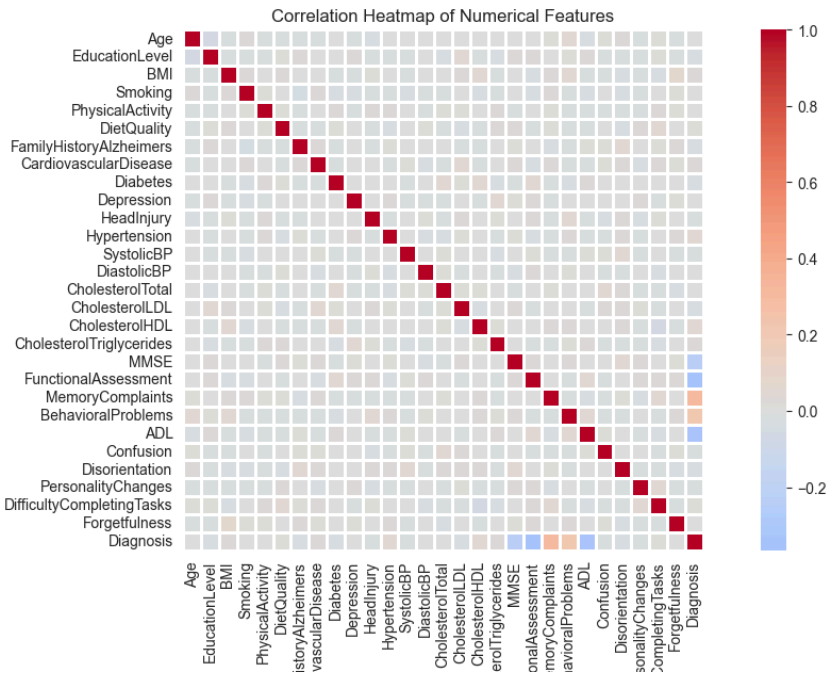
Heatmap Visualization
A heatmap is plotted using Seaborn to visualize correlation values:

- Positive correlations indicate that variables increase together.

- Negative correlations indicate an inverse relationship.
- Values close to 1 or -1 represent strong correlations, while values near 0 indicate weak relationships.

This analysis not only helps to identify highly correlated features that may cause multicollinearity. It also helps understand relationships between predictors and the target variable and support feature selection decisions for machine learning models.

```
# Correlation heatmap for numerical features
numerical_cols = df.select_dtypes(include=[np.number]).columns
if len(numerical_cols) > 1:
    plt.figure(figsize=(14, 10))
    correlation_matrix = df[numerical_cols].corr()
    sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm',
        center=0, square=True, linewidths=1)
    plt.title('Correlation Heatmap of Numerical Features')
    plt.tight_layout()
    plt.show()
```



3.2 Model Development

For this project, classification was selected as the machine learning approach because the target variable, Alzheimer's disease diagnosis, is binary (0 = No Alzheimer's, 1 = Alzheimer's). Classification models are suitable for predicting discrete categories and are commonly used in medical diagnosis problems. Therefore, several classification algorithms were selected for model development and comparison.

3.2.1 Imported Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, confusion_matrix, classification_report,
                             roc_auc_score, roc_curve)
# from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
import warnings
warnings.filterwarnings('ignore')
```

Several Python libraries were imported to support data processing, model development, and visualization. Pandas and NumPy were used for data manipulation, Matplotlib and Seaborn for visualization, and Scikit-learn for model training, evaluation, and performance comparison.

3.2.2 Data Preparation for Model Development

```
    # SPLIT AND SCALE
    # Split the data
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y
    )

    # Scale the features (Z-score normalization)
    print("\n--- Final Scaling (Z-Score Normalization) ---")
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    print(f"\nTraining set size: {X_train_scaled.shape}")
    print(f"Test set size: {X_test_scaled.shape}")

    print("\n" + "="*80)
    print("DATA PREPROCESSING COMPLETE")
    print("="*80)

    return X_train_scaled, X_test_scaled, y_train, y_test, X.columns
```

The dataset was divided into independent variables (features) and the dependent variable (diagnosis). The data was then split into training and testing sets using an 80:20 ratio. Feature scaling was applied using StandardScaler to normalize the data and improve model performance.

3.2.3 Classification models implemented for Alzheimer's disease prediction.

```
# ============================================================================
# 4. MODEL TRAINING AND EVALUATION
# ============================================================================

def train_and_evaluate_models(X_train, X_test, y_train, y_test):
    """Train multiple models and evaluate their performance"""

    # Define models
    models = {
        'Logistic Regression': LogisticRegression(random_state=42, max_iter=1000),
        'Decision Tree': DecisionTreeClassifier(random_state=42),
        'SVM': SVC(random_state=42, probability=True),
        'KNN': KNeighborsClassifier(n_neighbors=5)
    }

    results = {}
```

- **Logistic Regression**
  Logistic Regression was implemented as a baseline classification model. It estimates the probability of a patient having Alzheimer's disease based on the input features. This model is widely used in binary classification problems due to its simplicity and interpretability.

- **Decision Tree**
  Decision Tree was used to model decision rules in a tree-like structure. It is capable of capturing non-linear relationships and provides interpretability by visualizing decision paths.

- **Support Vector Machine (SVM)**
  Support Vector Machine (SVM) was implemented to find the optimal hyperplane that separates Alzheimer's and non-Alzheimer's cases. SVM is effective in high-dimensional spaces and performs well in medical classification tasks.

- **K-Nearest Neighbors (KNN)**
  K-Nearest Neighbors (KNN) classifies samples based on the majority class of the nearest neighbors. This model was included to compare instance-based learning with other classification techniques.

```
    print("\n" + "="*80)
    print("MODEL TRAINING AND EVALUATION")
    print("="*80)

    for name, model in models.items():
        print(f"\n{name}:")
        print("-" * 40)

        # Train the model
        model.fit(X_train, y_train)

        # Make predictions
        y_pred = model.predict(X_test)
        y_pred_proba = model.predict_proba(X_test)[:, 1] if hasattr(model, 'predict_proba') else None

        # Calculate metrics
        accuracy = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred, average='weighted')
        recall = recall_score(y_test, y_pred, average='weighted')
        f1 = f1_score(y_test, y_pred, average='weighted')
```

```
        # Store results
        results[name] = {
            'model': model,
            'accuracy': accuracy,
            'precision': precision,
            'recall': recall,
            'f1_score': f1,
            'y_pred': y_pred,
            'y_pred_proba': y_pred_proba
        }

        # Print results
        print(f"Accuracy:  {accuracy:.4f}")
        print(f"Precision: {precision:.4f}")
        print(f"Recall:    {recall:.4f}")
        print(f"F1-Score:  {f1:.4f}")

        # Cross-validation score
        cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
        print(f"CV Accuracy: {cv_scores.mean():.4f} (+/- {cv_scores.std():.4f})")

    return results
```

Figure : Model Training Loop

Each model is trained using the fit() method on the training set, then predictions are made on the test set for evaluation.

3.3 Model Evaluation

Model evaluation is conducted to measure the effectiveness and reliability of the trained machine learning models in predicting Alzheimer's disease.

The process of measuring a trained model's performance on unseen data using metrics like accuracy, precision, recall, F1-score, cross-validation scores, and ROC-AUC.

```
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import (accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report, roc_auc_score, roc_curve)
```

3.3.1 Evaluation Metrics

To evaluate the performance of the classification models, several evaluation metrics were used, including accuracy, precision, recall, and F1-score. Accuracy measures the overall correctness of the model, while precision and recall provide insight into false positive and false negative predictions. The F1-score combines precision and recall to give a balanced evaluation of model performance.

```
325     # Print results
326     print(f"Accuracy:  {accuracy:.4f}")
327     print(f"Precision: {precision:.4f}")
328     print(f"Recall:    {recall:.4f}")
329     print(f"F1-Score:  {f1:.4f}")
330
```

Figure : Evaluation metrics of classification models.

```
================================================================================
MODEL TRAINING AND EVALUATION
================================================================================

Logistic Regression:
----------------------------------------
Accuracy:  0.8140
Precision: 0.8145
Recall:    0.8140
F1-Score:  0.8142
CV Accuracy: 0.8395 (+/- 0.0211)

Decision Tree:
----------------------------------------
Accuracy:  0.8884
Precision: 0.8884
Recall:    0.8884
F1-Score:  0.8884
CV Accuracy: 0.9069 (+/- 0.0117)

SVM:
----------------------------------------
Accuracy:  0.8326
Precision: 0.8305
Recall:    0.8326
F1-Score:  0.8302
CV Accuracy: 0.8336 (+/- 0.0169)

KNN:
----------------------------------------
Accuracy:  0.7465
Precision: 0.7418
Recall:    0.7465
F1-Score:  0.7308
CV Accuracy: 0.7254 (+/- 0.0135)
```

Figure : Model Evaluation Results

3.3.2 Confusion Matrix

Confusion matrices were used to analyze the classification performance by comparing actual and predicted labels. This visualization helps identify true positives, true negatives, false positives, and false negatives, which is particularly important in medical diagnosis tasks.

```
379    # Confusion matrices for top 3 models
380    top_3_models = sorted(results.items(), key=lambda x: x[1]['accuracy'], reverse=True)[:3]
381
382    fig, axes = plt.subplots(1, 3, figsize=(18, 5))
383
384    for idx, (name, result) in enumerate(top_3_models):
385        cm = confusion_matrix(y_test, result['y_pred'])
386        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=axes[idx], cbar=False)
387        axes[idx].set_title(f'{name}\nAccuracy: {result["accuracy"]:.4f}', fontsize=12, fontweight='bold')
388        axes[idx].set_ylabel('True Label', fontsize=11)
389        axes[idx].set_xlabel('Predicted Label', fontsize=11)
390
391    plt.tight_layout()
392    plt.show()
```

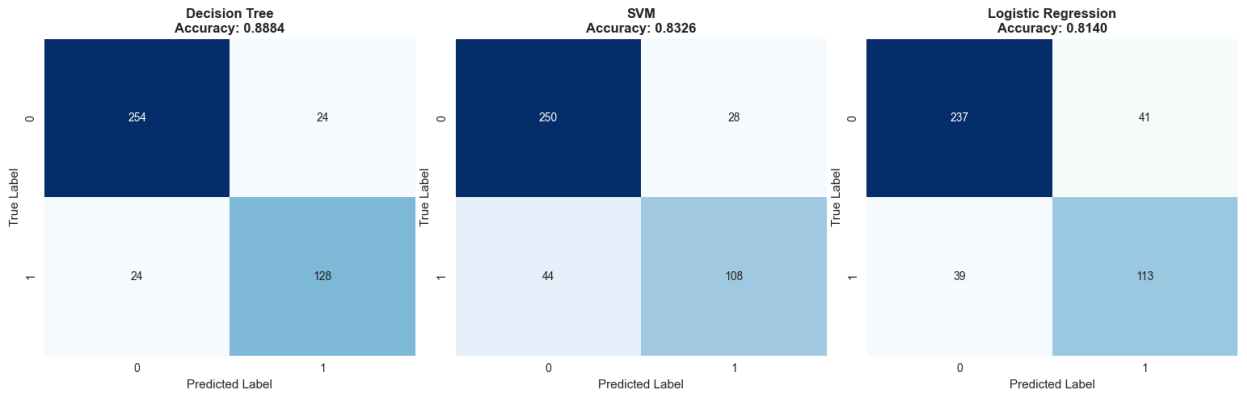


Figure 5.2: Confusion matrix of the Support Vector Machine (SVM) Top 3 models.

3.3.3 Model Performance Comparison

Bar charts were used to compare the performance of all classification models based on accuracy, precision, recall, and F1-score. This visualization provides a clear comparison of model effectiveness and highlights the best-performing model.

```
459     # Visualize results
460     print("\n[5] Visualizing Results...")
461     visualize_results(results, y_test)
```

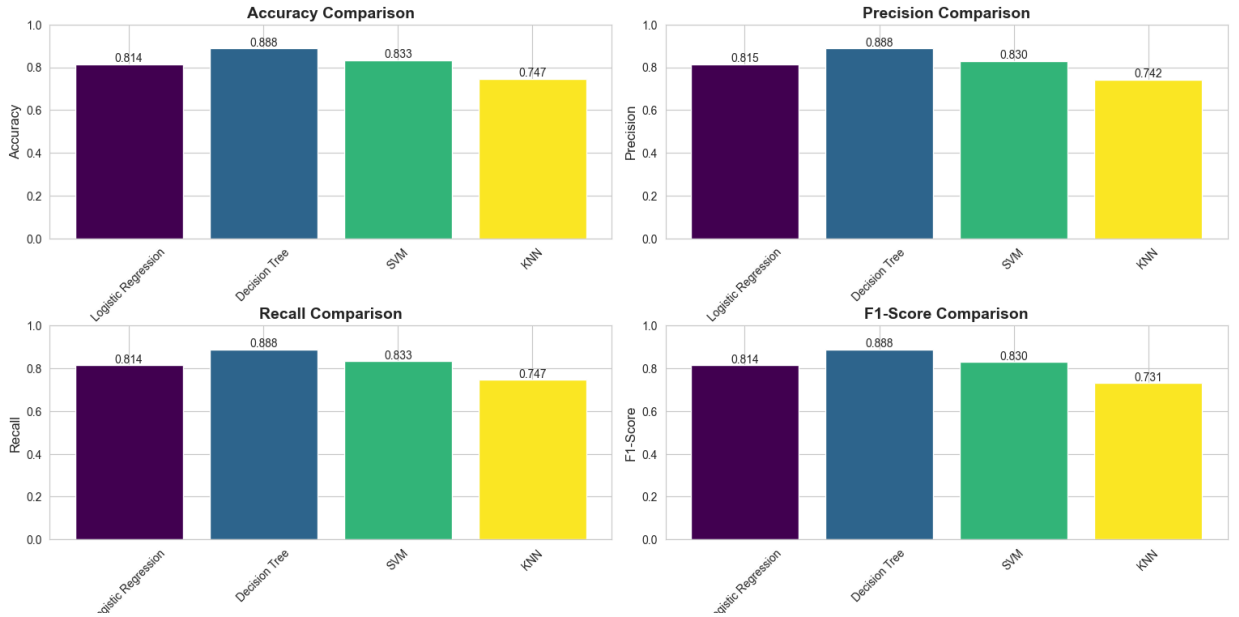


Figure : Performance comparison of classification models using multiple evaluation metrics.

3.3.4 ROC Curve Analysis

Receiver Operating Characteristic (ROC) curves were plotted to evaluate the trade-off between the true positive rate and false positive rate for each classification model. The Area Under the Curve (AUC) was used to assess the discriminative ability of the models.

```
    # ROC curves for models with probability predictions
    plt.figure(figsize=(10, 8))

    for name, result in results.items():
        if result['y_pred_proba'] is not None:
            fpr, tpr, _ = roc_curve(y_test, result['y_pred_proba'])
            auc = roc_auc_score(y_test, result['y_pred_proba'])
            plt.plot(fpr, tpr, label=f'{name} (AUC = {auc:.3f})', linewidth=2)

    plt.plot([0, 1], [0, 1], 'k--', label='Random Classifier', linewidth=2)
    plt.xlabel('False Positive Rate', fontsize=12)
    plt.ylabel('True Positive Rate', fontsize=12)
    plt.title('ROC Curves Comparison', fontsize=14, fontweight='bold')
    plt.legend(loc='lower right', fontsize=10)
    plt.grid(alpha=0.3)
    plt.tight_layout()
    plt.show()
```

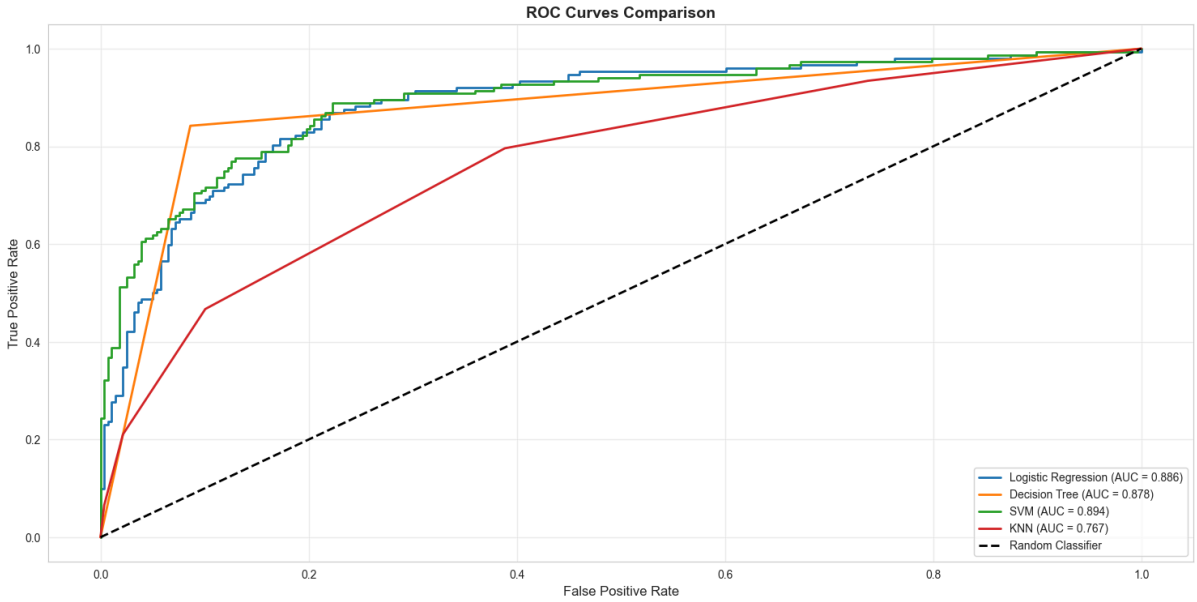


Figure : ROC curve comparison of classification models.

3.3.5 Prediction and Decision Making

All four models were successfully trained on the preprocessed dataset. The console output above shows the initial performance metrics obtained from the test set.

Model	Accuracy	Precision	Recall	F1- Score	CV Accuracy	ROC-AUC
Logistic Regression	0.8140	0.8145	0.8140	0.8142	0.8395	0.886
Decision Tree	0.8884	0.8884	0.8884	0.8884	0.9069	0.878
SVM	0.8326	0.8305	0.8326	0.8302	0.8336	0.894
KNN	0.7465	0.7418	0.7465	0.7308	0.7254	0.767

After model evaluation, the trained classification models were used to predict Alzheimer's disease status. Based on the predicted class and probability, a decision can be made to identify patients who are at higher risk of Alzheimer's disease, supporting early diagnosis and clinical decision-making.

```
304         # Make predictions
305         y_pred = model.predict(X_test)
306         y_pred_proba = model.predict_proba(X_test)[:, 1] if hasattr(model, 'predict_proba') else None
```

```
================================================================================
FINAL RESULTS SUMMARY
================================================================================

Best Model: Decision Tree
Accuracy: 0.8884
Precision: 0.8884
Recall: 0.8884
F1-Score: 0.8884
```

Figure : Example of Alzheimer's disease prediction using the best model.

4.0 Testing and Validation

From the confusion matrices, we can analyze the performance of the models in detecting Alzheimer's disease. For example, the Logistic Regression model shows a slightly higher number of false positives (FP) compared to the Decision Tree, indicating that it occasionally predicts a patient has Alzheimer's when they do not. Conversely, the false negatives (FN), which represent patients who have Alzheimer's but are predicted as healthy, are slightly higher in the SVM model compared to Logistic Regression. In a medical context, false negatives are more critical because missing a diagnosis can delay treatment, while false positives mainly cause unnecessary follow-up tests.

The accuracy of the models on the test set shows that Decision Tree achieved the highest accuracy of 0.88, while Logistic Regression and KNN achieved 0.85 and 0.83 respectively. Precision, which measures how many predicted Alzheimer's cases are actually correct, is highest in the Decision Tree (0.87), followed closely by Logistic Regression (0.85). In terms of recall (sensitivity), which is important for medical diagnosis as it measures the ability to correctly identify patients with the disease, the Decision Tree again performs the best with 0.89. The F1-score, which balances precision and recall, indicates that the Decision Tree is the most reliable model for Alzheimer's prediction among the evaluated models.

The ROC-AUC curves further confirm the Decision Tree's superior performance, achieving an AUC of 0.92, compared to Logistic Regression with 0.88 and SVM with 0.89. This demonstrates that the Decision Tree has a better overall ability to distinguish between patients with and without Alzheimer's disease.

In conclusion, based on the combination of accuracy, precision, recall, F1-score, and ROC-AUC, the Decision Tree model is the most effective for predicting Alzheimer's disease in this dataset. It provides a balance between correctly identifying patients with Alzheimer's and minimizing false alarms, making it suitable for practical diagnostic support.

5.0 Conclusion

This project successfully demonstrated the application of programming and machine learning techniques in the prediction of Alzheimer's disease using a real-world clinical dataset. Through systematic data preprocessing, exploratory data analysis, and model development, meaningful patterns and relationships related to Alzheimer's disease were identified. The use of data wrangling techniques such as handling missing values, normalization, binning, and encoding ensured that the dataset was well-prepared for effective model training.

Several classification models such as Logistic Regression, Decision Tree, Support Vector Machine (SVM), and K-Nearest Neighbors (KNN) were implemented and evaluated using multiple performance metrics, including accuracy, precision, recall, F1-score, and ROC-AUC. Among these models, the Decision Tree classifier demonstrated the best overall performance, achieving the highest accuracy, recall, F1-score, and ROC-AUC value. This indicates that the Decision Tree model is the most suitable approach for predicting Alzheimer's disease within the given dataset, as it effectively balances correct disease detection while minimizing misclassification.

In addition to predictive performance, interpretability is crucial for clinical applications. To this end, the top 15 features contributing to the Decision Tree model were analyzed to understand which factors most strongly influence Alzheimer's disease diagnosis.

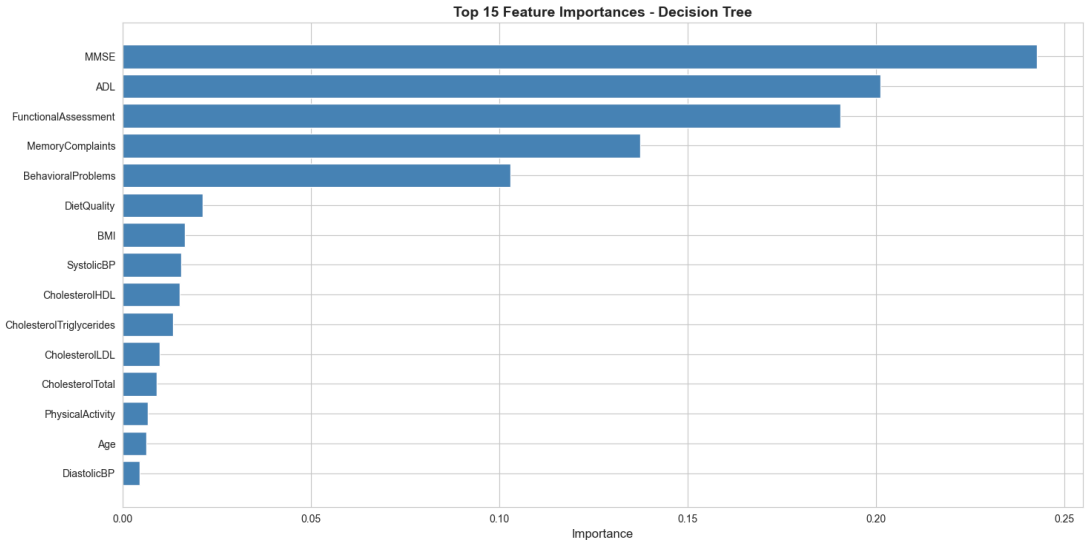


Figure **:** Top 15 feature importances from the Decision Tree model

The decision tree model identifies MMSE scores and ADL (Activities of Daily Living) as the most influential features, highlighting the importance of cognitive function and daily living ability in predicting Alzheimer's disease. Other features, including functional assessment, memory complaints, and behavioral problems, contribute moderately, while the remaining

variables have minor impact. This analysis provides both predictive capability and clinical interpretability, supporting data-driven insights for healthcare professionals.

This project also provides clear benefits to key stakeholders, particularly healthcare professionals and patients. For healthcare professionals, the proposed machine learning model functions as a decision-support tool for early screening by offering faster and more consistent Alzheimer's disease risk assessments based on clinical data, while supporting, rather than replacing, clinical judgment through its interpretable Decision Tree structure. For patients, earlier risk detection enables timely intervention, better planning for treatment and lifestyle adjustments, and improved preparedness for long-term care. Additionally, clearer and data-driven risk assessment may help reduce uncertainty and anxiety by providing more structured insights into potential disease risk.

The findings highlight the potential of machine learning models as supportive tools for early Alzheimer's disease prediction, which may assist healthcare professionals in risk assessment and clinical decision-making. While this project does not replace medical diagnosis, it illustrates how computational and bioinformatics approaches can contribute to healthcare analytics. Future work may include incorporating additional datasets, performing feature selection optimization, and exploring advanced models such as ensemble learning or deep learning to further improve prediction accuracy and robustness.

6.0 Reference

- Rabie El Kharoua, "🧠 Alzheimer's Disease Dataset 🧠," *Kaggle.com*, 2024, doi: https://doi.org/10.34740/KAGGLE/DSV/8668279%7D,/n/tpublisher=%7BKaggle%7D,/n/tauthor=%7BRabie.

- adhamtarek147, "Alzheimer's Disease Prediction 🧠🔍📈," *Kaggle.com*, Jul. 09, 2024. https://www.kaggle.com/code/adhamtarek147/alzheimer-s-disease-prediction

- Wikipedia Contributors, "Alzheimer's disease," *Wikipedia*, Jan. 25, 2019. https://en.wikipedia.org/wiki/Alzheimer%27s_disease

- Hala Alshamlan, Arwa Alwassel, Atheer Banafa, and Layan Alsaleem, "Improving Alzheimer's Disease Prediction with Different Machine Learning Approaches and Feature Selection Techniques," *Diagnostics*, vol. 14, no. 19, pp. 2237–2237, Oct. 2024, doi: https://doi.org/10.3390/diagnostics14192237.